
# Using OpenMP 4 on Pleiades

July 26, 2017
NASA Advanced Supercomputing Division

# Outline

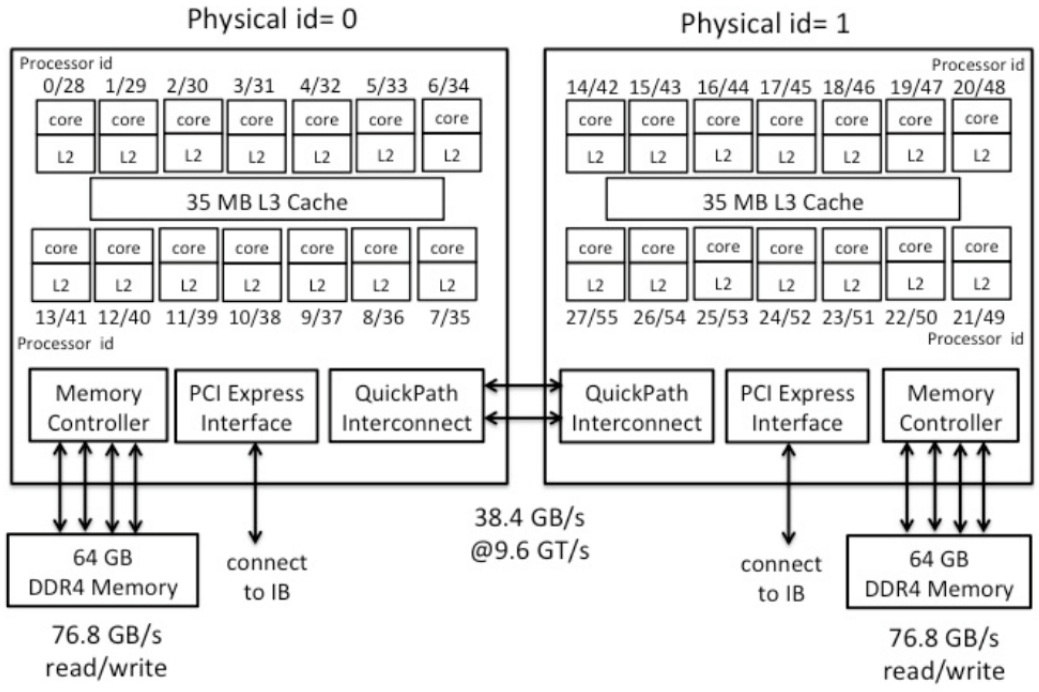
- Introduction
  - Pleiades Node Architecture
- Review of OpenMP 3 Basics
  - Parallel constructs and data sharing
  - Work-sharing constructs
  - Synchronization and other important constructs
- Vectorization with OpenMP 4
- Compiling and Running OpenMP Codes
- Compiling and Running MPI+OpenMP Codes



Question? Use the Webex chat facility to ask the Host

# Pleiades Node Architecture

- Cluster of shared memory multi-core processors:
  - ~11400 nodes, total of 246,048 cores
  - Multiple processors per node
  - Multiple cores per processor
  - No data sharing among the nodes
  - Various levels of data sharing within a node (memory, caches)




Xeon Broadwell

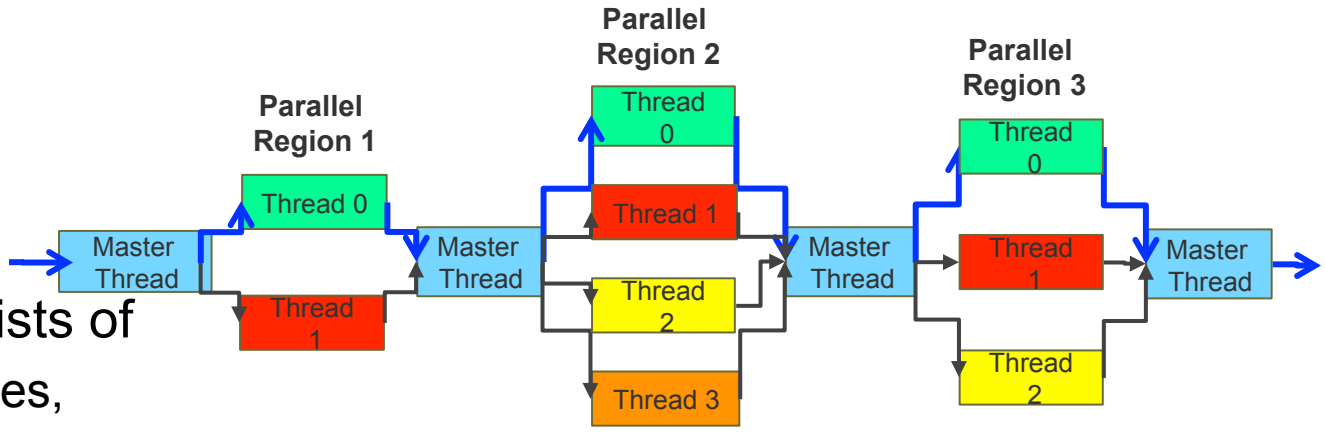
There is a potential shared memory parallelism up to 56 threads.



Question? Use the Webex chat facility to ask the Host

# What is OpenMP?

- An API for Shared Memory Programming
- OpenMP Thread
  - Execution engine with local memory and access to the shared memory
- Fork-Join Execution Model:
  - Threads are dynamically created and managed by the OpenMP runtime
- Major OpenMP API components:
  - Parallelization
  - Data Sharing
  - Work Sharing
  - Synchronization
- OpenMP API consists of
  - Compiler directives,
  - Runtime library routines
  - Environment variables




Fortran: !$omp

C/C++: #pragma omp



Question? Use the Webex chat facility to ask the Host

# Parallelization, Data Sharing, Work Sharing

- **Parallelization**:
  - **`parallel:`**
    - Threads are being forked
    - All threads execute same code
- **Important Data Sharing Clauses**
  - **`shared`**: default
  - **`private:`** local to a thread
  - **`reduction:`** values calculated across all threads, e.g. a global sum
  - **`firstprivate,lastprivate`**, etc
  - Some default data scoping rules, but if in doubt use data sharing clause
  - A good practice: **`default (none)`** (sometimes not feasible)
- **Work Sharing**
  - **`do`** (Fortran) / **`for`** (C): loop iterations
  - **`sections`**: code blocks

```
!$omp parallel do num_threads(4)
 do i = 1, n
    do j = 1, n
       a (i) = b (i, j) + 5.
     end do
   end do
!$omp end parallel do
```

```
#pragma omp parallel for num_threads(4)
 for (i = 1; i<=n; i++){
      for (int j = 1; j <= n; j++) {
         a (i) = b (i, j) + 5.
  } }
```

**Fortran**: i and **j** are private by default
**C**: i is private, **j** is shared by default. It needs to be declared or declared within the parallel region, as in the example

How many threads are working?

**`setenv OMP_NUM_THREADS nt`**

- Runtime library call overwrites env variable

**`omp_set_num_threads (nt)`**

- Clause is the strongest

**`omp parallel num_threads (nt)`**

# Scheduling the Work

- Who is doing what?
- `omp for/do schedule(static, chunk-size)`:
  - Loop chopped up into approximately equal blocks
  - Each thread assigned a block of iterations
  - Lowest overhead
  - Default for most compilers
  - Good if the workload is balanced
- `omp for/do schedule (dynamic, chunk-size)`
  - Threads request chunks until no more are left
  - More overhead than static
  - Better load-balance if the work per chunk varies

  •Others: `guided, auto`

```
!$omp parallel
!$omp do schedule(dynamic)
   do i = 0, n
      call sub1(a, b, c, i)
   end do
!$omp end do
!$omp end parallel
```



Question? Use the Webex chat facility to ask the Host

# Other Important Constructs

- **Synchronization**
  - `barrier` explicitly synchronizes all threads in a team
    - Removable implicit barriers at the end of work sharing constructs (`nowait` clause)
    - Non-removable implicit barriers at the the end of parallel construct
  - `critical`
    - Region of code accessed by one thread at a time
  - `atomic`:
    - Memory location updated atomically
    - Faster than critical, if applicable
  - Others: `flush,ordered`
- **Another type of work-sharing**: One does the work, the others sleep
  - `single` … or spin
    - Executed by a single thread
    - Implicit barrier
- **Some things should only be done by the boss**
  - `master`
    - Executed only by the master thread
    - No implicit barrier

```
!$omp parallel do
  do i =1,npt
     …
!$omp critical
    call lib_sub1(t_shared)
!$omp end critical

     …
     end do
!$omp end parallel do
```

```
subroutine lib_sub1 (is)
 logical first
 save first
 if (first) then
    first = .false.
    do_stuff (is)
 endif
 return
end
```

**Beware of non-threadsafe library calls**:
e.g. write to a global variables by multiple threads



Question? Use the Webex chat facility to ask the Host

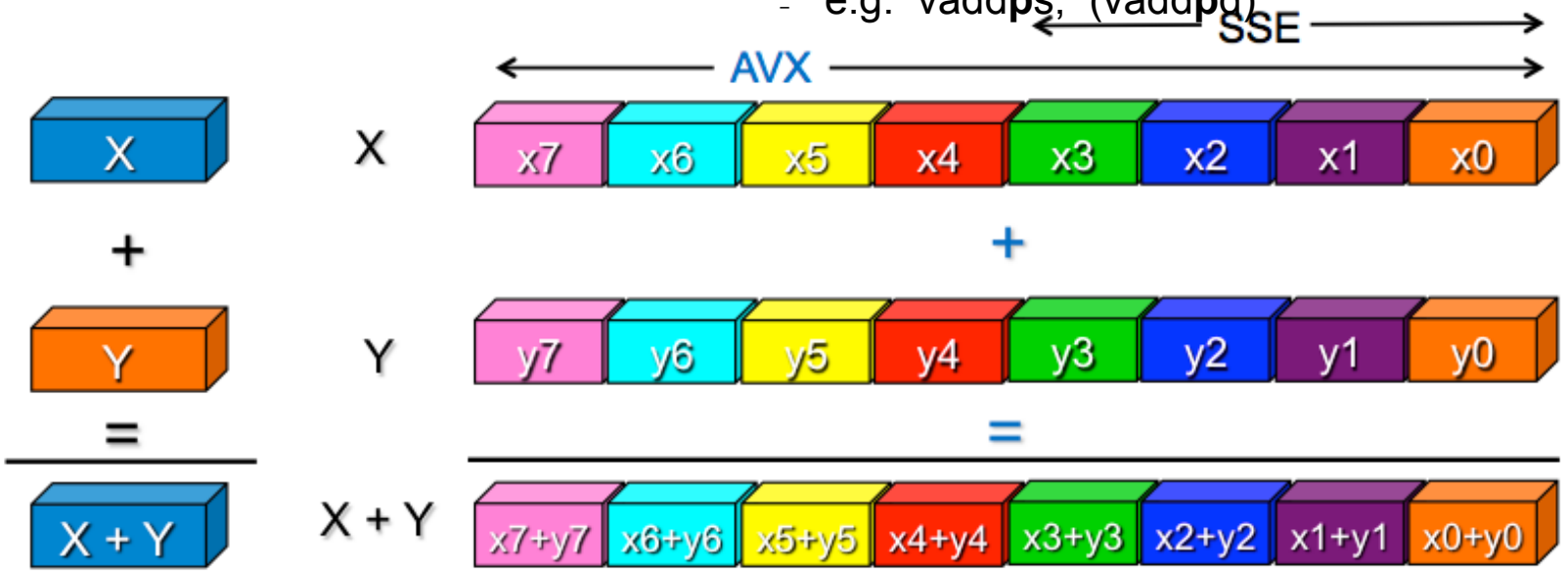
# What is Vectorization?

- Execute a **S**ingle **I**nstruction on **M**ultiple **D**ata

```
do i = 1, n
        a(i) = x(i) + y(i)
end do
```

```
for (i=0; i<n+1;i++)
    a[i]  = x[i] + y[i]
```

- Scalar mode
  - one instruction produces one result
  - e.g. vadd**s**s, (vadd**s**d)
- Vector mode
  - one instruction can produces multiple results
  - e.g. vadd**p**s, (vadd**p**d)




Note: Image borrowed from Intel Tutorial!

4 floats in SSE
8 floats for AVX
16 floats for AVX512



Question? Use the Webex chat facility to ask the Host

# Vectorization with OpenMP 4

- It is not always possible for the compiler to vectorize a loop, due to assumed dependences
- Vendors provided directives/pragmas for loop vectorization
- OpenMP 4.0 provides a standardization for vectorization via the SIMD construct
  - The loop is subdivided into chunks that fit in a SIMD vector (strip-mining)

```
#pragma omp simd <clauses>
#pragma omp declare simd <clauses>
```

to vectorize loops

vector routines

```
void addit (double* a, double *b, int n, int off)

#pragma omp simd ?
    for (int i = 0; i < n; i++ {
        a [i] = b [i] + a [i – off]
    }
```

off <= 0 **ok**
off > 0 **might be incorrect!!**

# Golden Rules for OpenMP 4 SIMD Constructs

- Don't use them. Let the compiler vectorize, if possible:
  - Prefer simple "for" or "do" loops over "while"
  - Disambiguate function arguments, via compiler flags eg,
  - `-fargument-noalias` or `-restrict` for Intel compilers
  - Inspect optimization reports (Intel) to find obstacles
- OpenMP simd directives are commands to the compiler, not hints:
  - The user is responsible for correctness!
  - Use data scoping clauses as necessary
- **Only use as last resort!**
  - Time consuming loops are not vectorized by compiler
  - Optimization report shows "false" assumed dependences
- Vectorization may change the results, e.g. reduction operations, transcendental functions, others.

**YOU are responsible for correctness!**

# Example: C++ SIMD Vectorization

```
....
 for(int k=0; k<ncells3; ++k){
        for(int j=0; j<ncells2; ++j){
          for(int i=0; i<ncells1; ++i){
             Real vx = w(IVX,k,j,i);
             Real vy = w(IVY,k,j,i);
             Real vz = w(IVZ,k,j,i);
             for(int ifr=0; ifr<nfreq; ++ifr){
               Real ds = pco->dx1v(i);
               ….
   #pragma omp simd
               for(int n=0; n<nang; ++n){
                 Real vdotn = vx*prad->mu(0,k,j,i,n)+vy*prad->mu(1,k,j,i,n)
                            + vz*prad->mu(2,k,j,i,n)
                 vdotn *= invcrat
                 Real adv_coef = tau_fact * vdotn * (3.0 + vdotn * vdotn);
                 Real q1 = ir(k,j,i,n+ifr*nang) * (1.0 - adv_coef);
                 temp_i1(k,j,i,n+ifr*nang) = q1;
                 temp_i2(k,j,i,n+ifr*nang) = adv_coef
               }}}}}
```

User confirmed: No overlap of `prad` and temp_i2:

Ok to use simd !

```
 LOOP BEGIN at src/radiation/integrators/rad_transport.cpp(111,16)
                remark #15344: loop was not vectorized: vector dependence
prevents vectorization
                remark #15346: vector dependence: assumed ANTI dependence
between prad line 116 and temp_i2.temp_i2 line 121
LOOP END
```



Question? Use the Webex chat facility to ask the Host

# Example: C++ SIMD Reduction

```
    ….
    Real er0 = 0.0;
    …
        for(int ifr=0; ifr<nfreq; ++ifr){
#pragma omp simd reduction (+:er0)
           for(int n=0; n<nang; ++n){
             Real ir_weight = lab_ir[n+ifr*prad->nang];

             er0 += ir_weight;
              ..
           }
         er0 *= prad->wfreq(ifr);
      }
```

shared variable, updated to hold an accumulated sum => use reduction clause



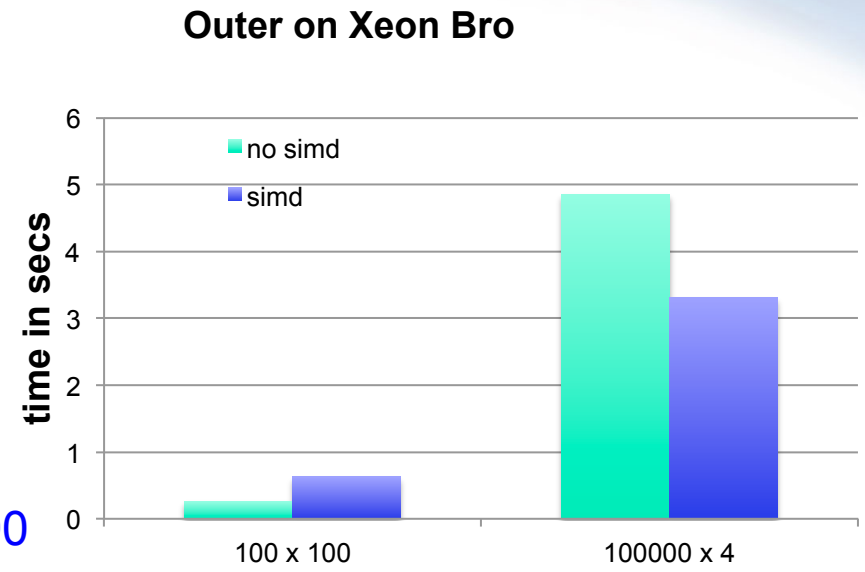
Question? Use the Webex chat facility to ask the Host

# Example: SIMD for Outer Loop Vectorization

```
!$omp simd private(d)
 do i = 1, n
    d = 0.
    do j = 1, nd
       d = d + (a( j, i) - b (j) ) ** 2
    end do
    dis (i) = sqrt (d)
 end do
```

Outer loop with high trip count

Inner loop with low trip count




ifort –c –qopt-report=5 –xcore-avx2 outer.f90

```
LOOP BEGIN at outer.f90(19,8)
   remark #15542: loop was not vectorized: inner loop was already vectorized
LOOP BEGIN at outer.f90(21,11)
remark #15300: LOOP WAS VECTORIZED
```

ifort –c –qopenmp-simd –qopt-report=5 –xcore-avx2 outer.f90

```
LOOP BEGIN at outer.f90(19,8)
remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
 LOOP BEGIN at outer.f90(21,11)
    remark #15548: loop was vectorized along with the outer loop
```



Question? Use the Webex chat facility to ask the Host

# Example: SIMD Enabled Subroutine

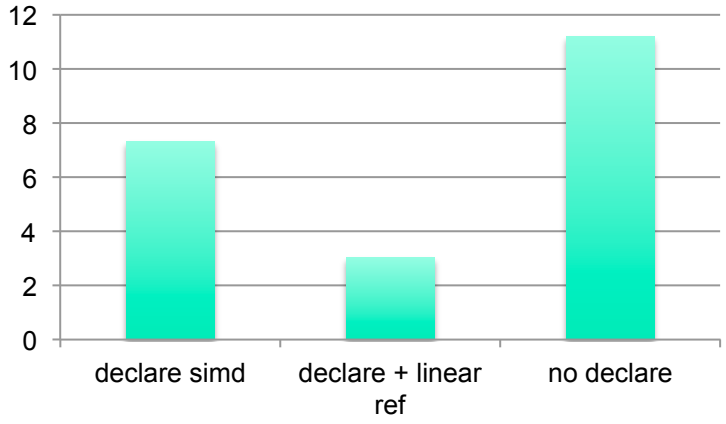
- Compiler generates SIMD-enabled (vector) version of a scalar subroutine that can be called from a vectorised loop

```
subroutine test_linear(x, y)
!$omp declare simd (test_linear) linear(ref(x, y))
real(8),intent(in)  :: x
    real(8),intent(out) :: y
    y = 1. + sin(x)**3
end subroutine test_linear
```

OpenMP 4.5

```
!$omp simd
do i = 1, n
    …
   call test_linear (a(i), b(i))
end do
```

**Time in secs on Xeon Bro**




remark #15347: FUNCTION WAS VECTORIZED with ymm2, simdlen=4,
remark #15415: vectorization support: indirect load was generated for the variable <x>
remark #15329: vectorization support: indirect store was emulated for the variable <y>
remark #15462: **unmasked indexed (or gather) loads: 1**
**remark #15463: unmasked indexed (or scatter) stores:** 1

**SLOW!**

**with linear(ref)**

remark #15347: FUNCTION WAS VECTORIZED with ymm2, simdlen=8, unmasked,
**remark #15450: unmasked unaligned unit stride loads: 1**
**remark #15451: unmasked unaligned unit stride stores: 1**

**Fast**



**Question? Use the Webex chat facility to ask the Host**

# Compiling OpenMP Codes

- Intel icc/ifort

```
pfe27> module load comp-intel
pfe27> module list
Currently Loaded Modulefiles:
  1) comp-intel/2016.2.181

icc/ifort –qopenmp –c test.c/.f

icc/ifort –qopenmp-simd –c test.c/.f
```

Add –qopt-report=5 for optimization report

- Gnu gcc/gfortran

```
pfe27 > module load gcc
pfe27 > module list
Currently Loaded Modulefiles:
  1) gcc/6.2

gcc/gfortran –fopenmp –c test.c/.f

gcc/gfortran –fopenmp-simd –c test.c/.f
```

"omp simd" only,
No omp parallel
No OpenMP runtime

# Running OpenMP Codes on Pleiades

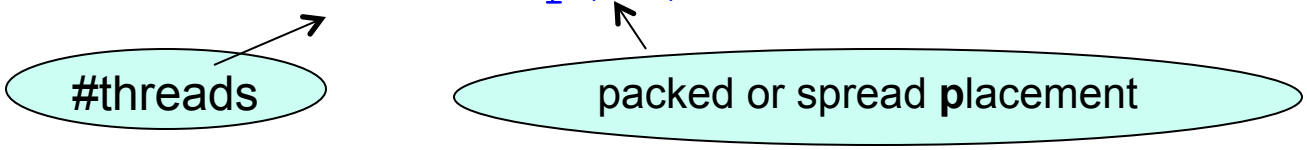
- Using Intel KMP or OMP Affinity environment variables for thread placement

```
setenv OMP_NUM_THREADS 8
setenv OMP_PROC_BIND spread
setenv OMP_PLACES cores
./test.x
```

Do not mix the thread placement methods, one never knows how they play with each other!

- Using tools, e. g, **mbind.x**

```
setenv OMP_NUM_THREADS 8
mbind.x -t<n> -c<p,s,..>./test.x
```



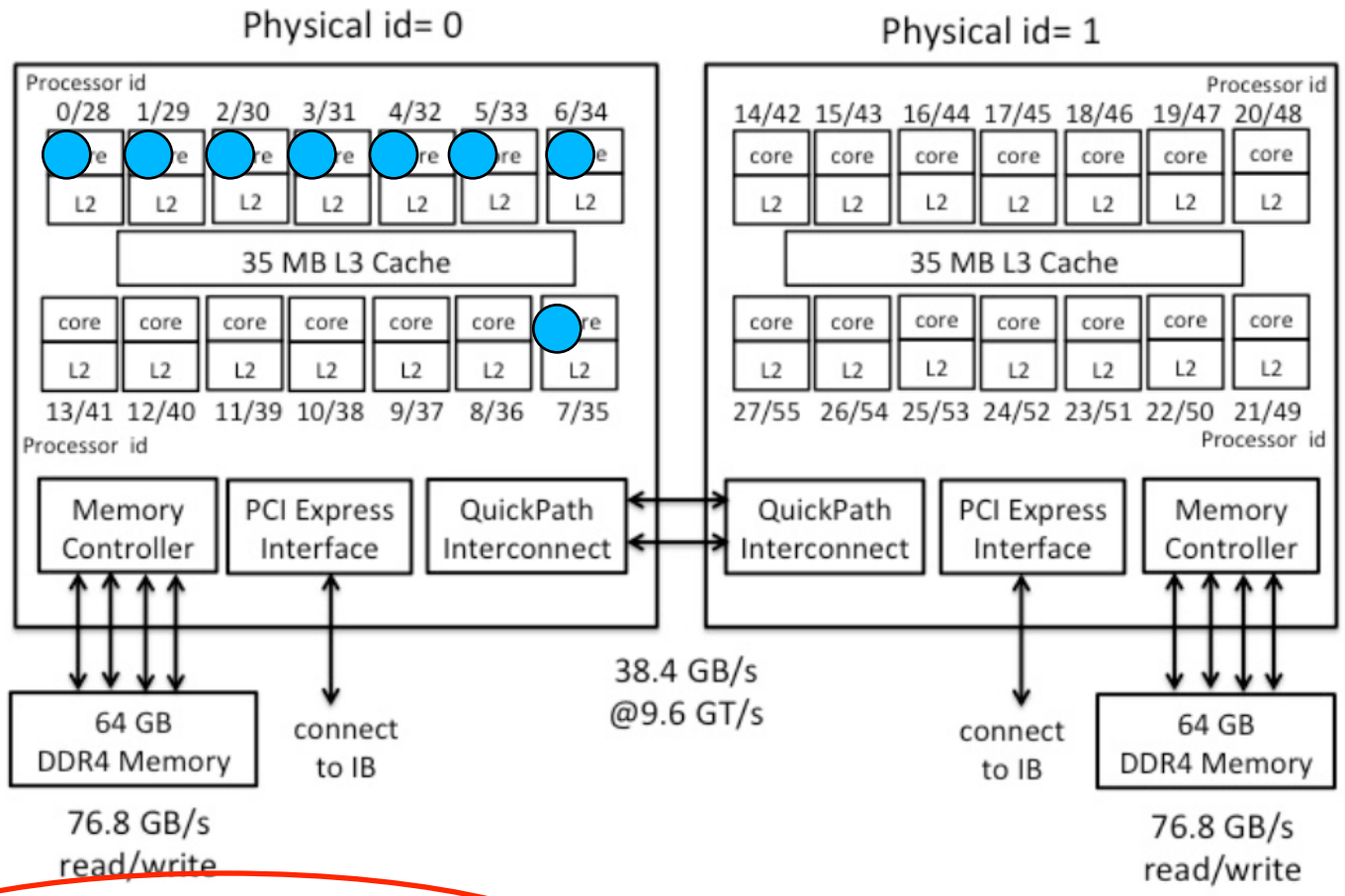

- Calls to runtime libraries:

```
sched_setaffinity, sched_getaffinity
```



Question? Use the Webex chat facility to ask the Host

# Thread Placement Examples



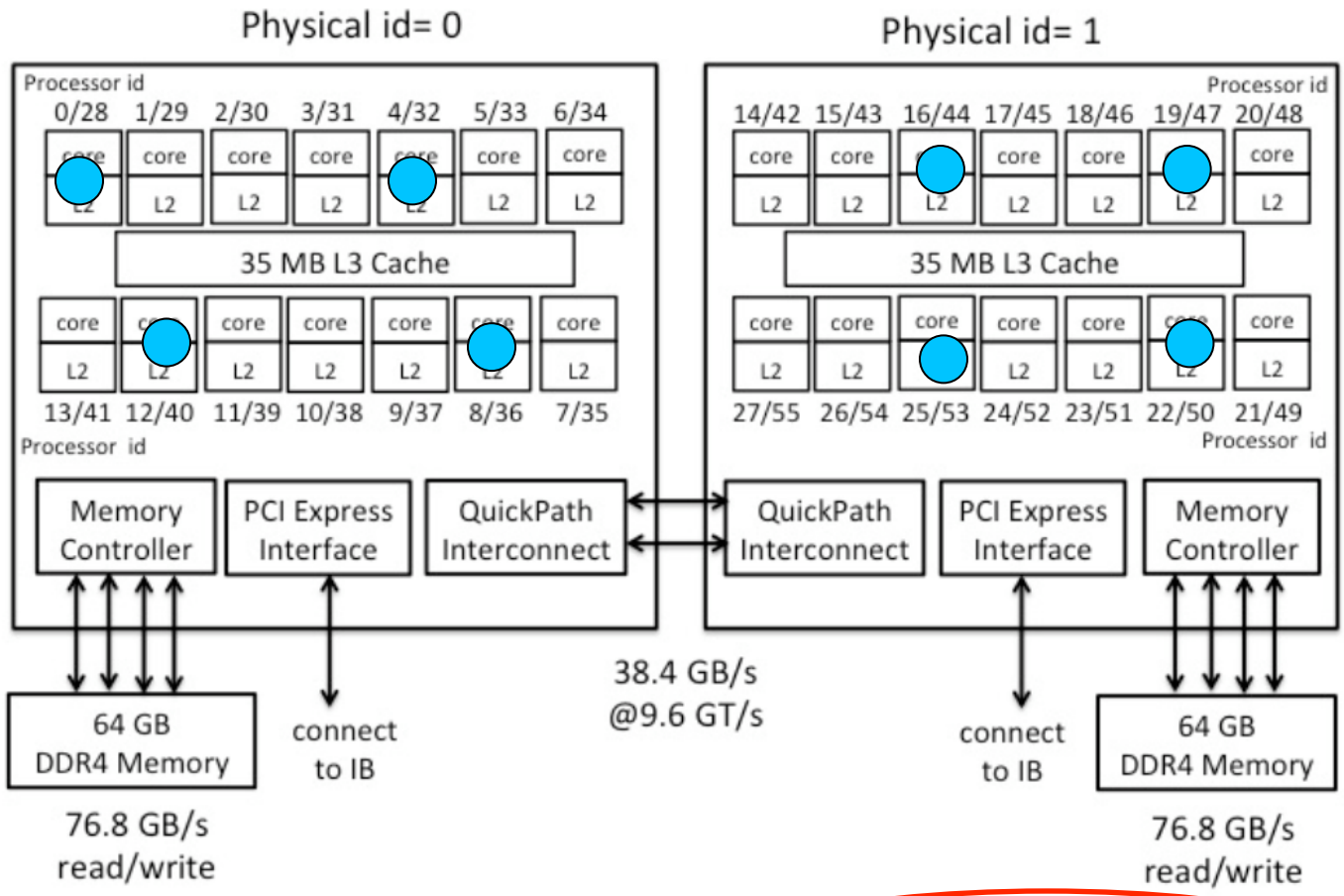

```
setenv OMP_PROC_BIND close
setenv OMP_PLACES cores
```

```
setenv OMP_PROC_BIND spread
setenv OMP_PLACES cores
```



Question? Use the Webex chat facility to ask the Host

# Thread Placement Examples




```
setenv OMP_PROC_BIND close
setenv OMP_PLACES cores
```

```
setenv OMP_PROC_BIND spread
setenv OMP_PLACES cores
```



Question? Use the Webex chat facility to ask the Host

# Compiling and Running Hybrid Codes

- Compilation

```
module load mpi-sgi/mpt comp-intel
env MPICC_CC=icc mpicc -o th.x -O3 -qopenmp th.f90
```

by default gcc

- Enable SGI MPI for running hybrid codes

```
setenv MPI_DSM_DISTRIBUTE
setenv MPI_OPENMP_INTEROP
```

- Set the number of threads to be used

```
setenv OMP_NUM_THREADS 4
```

- Run the executable

```
mpiexec –np 56 ./th.x
```

**alternatively,** use mbind.x or omplace for process binding

- Request sufficient resources via PBS

```
qsub -l select=8:ncpus=28:mpiprocs=7:model=bro
```

alternatively generate new PBS_NODEFILE



Question? Use the Webex chat facility to ask the Host

# Summary

- OpenMP is a compiler directive based shared memory programming API
- Provides an easy way to parallelize time consuming loops within one Pleiades node
  - Multithreading done by the compiler behind the scenes
- Care has to be taken to synchronize access to shared data
  - User's responsibility
  - Debugging may be hard
- Vectorization with OpenMP SIMD requires great care and understanding of the hardware architecture
- Other OpenMP related possible topics:
  - Hybrid MPI+OpenMP details
  - Optimizing and debugging OpenMP
  - OpenMP tasking,
  - Off-loading to co-processors with OpenMP or OpenACC

**As you like it.**

Let us know!



Question? Use the Webex chat facility to ask the Host

# References

- Running OpenMP codes on Pleiades

https://www.nas.nasa.gov/hecc/support/kb/OpenMP_209.html

https://www.nas.nasa.gov/hecc/support/kb/porting-with-openmp_103.html

- Thread and Process Placement

https://www.nas.nasa.gov/hecc/support/kb/ProcessThread-Pinning-Overview_259.html

- Running hybrid codes on Pleiades

https://www.nas.nasa.gov/hecc/support/kb/With-SGIs-MPI-and-Intel-OpenMP_104.html

- OpenMP Specification

http://www.openmp.org/specifications/

- OpenMP Training Material

http://www.openmp.org/resources/tutorials-articles/

https://computing.llnl.gov/tutorials/openMP/

- OpenMP SIMD Vectorization

http://primeurmagazine.com/repository/PrimeurMagazine-AE-PR-12-14-32.pdf

http://www.hpctoday.com/hpc-labs/explicit-vector-programming-with-openmp-4-0-simd-extensions/



Question? Use the Webex chat facility to ask the Host